UNIVERSIDADE LUTERANA DO BRASIL

DIRETORIA ACADÊMICA

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PROJETO DE DESENVOLVIMENTO DE SOFTWARE

DESENVOLVIMENTO DE UMA WEB API COM .NET E ENTITY FRAMEWORK CORE

MONALISE SCHWANCK PINHO

LUCAS FOGAÇA

Torres, 2025

INTRODUÇÃO

Este artigo tem como objetivo descrever detalhadamente o processo de desenvolvimento de uma Web API. O sistema foi implementado utilizando a linguagem C#, o framework ASP.NET Core, e o Entity Framework Core como ORM, com persistência de dados em SQLite.

A aplicação busca facilitar o controle de tutores e seus respectivos pets, permitindo operações de cadastro, consulta, atualização e exclusão. O desenvolvimento seguiu as boas práticas de engenharia de software, como o uso do padrão Repository e a separação de responsabilidades.

DESENVOLVIMENTO

1. Requisitos Funcionais
    - **Cadastrar, atualizar, excluir e consultar tutores:**

    Essas funcionalidades foram implementadas no TutorController, que disponibiliza os métodos POST, PUT, DELETE e GET. Toda a lógica de acesso ao banco foi abstraída em uma interface ITutorRepository, com implementação concreta em TutorRepository, utilizando AppDbContext para persistência com Entity Framework Core.

    - Cadastrar, atualizar, excluir e consultar pets:

      Da mesma forma, o PetController implementa as rotas REST correspondentes às operações CRUD para pets. O repositório IPetRepository encapsula a lógica de banco, garantindo que a controller permaneça desacoplada da infraestrutura.

    - Cada pet pertence a um tutor específico:

      O modelo Pet possui uma propriedade TutorId como chave estrangeira. No DbContext, essa relação foi configurada com a anotação

[ForeignKey("TutorId")], e com a navegação via Tutor no modelo Pet e List<Pet> no modelo Tutor.

- **Validação de campos obrigatórios:**
  Nos modelos Tutor e Pet, os campos Nome e Telefone foram marcados com [Required]. A API retorna erros de validação automaticamente quando esses campos não são fornecidos.

2. Requisitos Não Funcionais

- **Persistência de dados com EF Core e SQLite:**
  O banco de dados foi configurado no AppDbContext e registrado no Program.cs com a string de conexão apontando para um arquivo SQLite. O Migrations foi utilizado para gerar a estrutura do banco
- **Código disponível em repositório GitHub público:**
  O projeto foi hospedado em um repositório público com nome e estrutura organizados. A pasta Artigo contém o presente documento em PDF, conforme exigência.
- **Boas práticas de desenvolvimento:**
  Foram seguidos os princípios SOLID, com uso do padrão Repository, injeção de dependência (AddScoped para os repositórios) e separação clara entre camadas.

CONCLUSÃO

O desenvolvimento da aplicação proporcionou a aplicação prática dos conhecimentos adquiridos em engenharia de software, orientação a objetos e arquitetura de APIs REST.

Entre os principais desafios:

- Configuração do relacionamento entre as entidades com EF Core.
- Separação de responsabilidades entre controllers e repositórios.
- Aplicação correta de validações.

Como principais aprendizados: a importância da organização do código, a clareza da separação de camadas e o uso eficiente do Entity Framework Core como ferramenta de mapeamento objeto-relacional.

Referências Bibliográficas

- MICROSOFT. ASP.NET Core documentation. Microsoft Docs, 2024.
- FREEMAN, Adam; ASP.NET Core in Action. Manning Publications, 2021.
- RICHTER, Jeffrey. CLR via C#. Microsoft Press, 2019.
- ALUR, Rajeev; HENZINGER, Thomas; Software Design and Engineering. MIT Press, 2020.